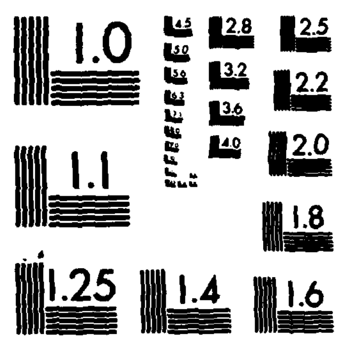
AD-A140 886 DESIGN METHODOLOGY FOR REAL-TIME DISTRIBUTED SYSTEMS (U) TECHNION - ISRAEL INST OF TECH HAIFA DEPT OF COMPUTER SCIENCE M YOELI DEC 83 TR121-597-003 1/1

UNCLASSIFIED RADC-TR-83-272 AFOSR-81-0152 F/G 9/2 NL

END



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A140 886

(1.7)

RADC-TR-83-272
Final Technical Report
December 1983

DESIGN METHODOLOGY FOR REAL-TIME DISTRIBUTED SYSTEMS

Technion-Israel Institute of Technology

Michael Yoeli

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, NY 13441

This report has been reviewed by the RADC Public Affairs Office (PA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

RADC-TR-83-272 has been reviewed and is approved for publication.

APPROVED:

FREDERICK A. NORMAND
Project Engineer

APPROVED:

RONALD S. RAPOSO, Acting Chief
Command and Control Division

FOR THE COMMANDER:

JOHN A. RITZ
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (COTC) Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return copies of this report unless contractual obligations or notices on a specific document requires that it be returned.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER RADC-TR-83-272	2. GOVT ACCESSION NO. AD-A140 886	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) DESIGN METHODOLOGY FOR REAL-TIME DISTRIBUTED SYSTEMS		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report 1 June 81 - 31 May 83 6. PERFORMING ORG. REPORT NUMBER TR121 - 597 - 003
7. AUTHOR(s) Michael Yoeli		8. CONTRACT OR GRANT NUMBER(s) AFOSR 0152-81-003
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Department Technion - Israel Institute of Technology Technion City, Haifa 32 000, Israel		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61102F 2304J408
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (COTC) Griffiss AFB NY 13441		12. REPORT DATE December 1983 13. NUMBER OF PAGES 44
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) EOARD/LNI, Box 14 FPO New -k 09510		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Frederick A. Normand (COTC)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Distributed Systems
Real-Time Systems
Structured Design

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report establishes a methodology for the structured design of complex, real-time digital systems, involving a high degree of concurrency. The design is based on the initial decomposition of the system specification into a control part and a data processing part. Formal models are developed for both parts, and a design methodology closely related to structured programming, is shown to be applicable. The proposed methodology is particularly suitable for precise and concise system require-

ments specifications as well as for the application of advanced verification methods.

TABLE OF CONTENTS

1. INTRODUCTION

This report is concerned with establishing a methodology for the design of complex real-time digital systems. These systems are dedicated to a single objective, such as flight-guidance, communication switching, patient monitoring, or industrial process-control. The overall task of the system can be decomposed into several subsidiary tasks, each of which contributes to the overall objective.

Efficient implementations exploit, as much as possible, the high degree of concurrency usually involved in such systems. Multimicrocomputer and VLSI implementations are of particular interest. Structured programming [LI-MI-WI] has become a generally accepted approach in modern software engineering. A similar approach can be applied to the design of complex, combined hardware/software systems, leading to a structured design methodology. The importance of such a design methodology has recently been emphasized, particularly in connection with the growing trend towards computer-aided design of VLSI-systems [LEW], [MEA-CON].

The major steps involved in a structured, top-down design approach are the following:

(1) system requirements specification

(2) stepwise refinement

(3) implementation

(4) verification.

In the following section we survey some of the publications dealing with the above design steps. In Sections 3-6 we develop an alternative methodology of specifying system requirements. In Section 7 we very briefly indicate the applicability of this method to the derivation of efficient and correct implementations.

2. SURVEY OF RELATED WORK

The difficulties involved in designing and maintaining complex software have led to extensive studies of suitable methodologies. In particular, the problem of software requirements specification has received considerable attention. Consequently, a variety of requirements specification languages have recently been developed. Typical examples of such languages are described in [DAV], [LEV-MUL], [ZAV]. These languages are mainly intended to facilitate the development of software, rather than hardware systems or combined hardware/software systems. They assume a well-defined, fixed architecture, for which a particular software is to be developed.

However, an essential advantage of any suitable structured system design is the integrated approach to hardware and software, enabling the designer to postpone his decision about hardware/software partitioning to a late stage in his design. Such a structured system design methodology calls for requirement specification methods applicable to both hardware systems as well as combined hardware/software systems. Of especial interest are specification methods which clearly establish feasible concurrences in the system.

Various research groups have recently devoted considerable efforts to the development of specification methods for complex, highly-concurrent systems, based on suitable modifications and extensions of the concept of Petri net. Some of these efforts are described in [VAL-COU], [MOA-DAV], [QUE], [WOJ], [YOE 82a], [YOE-BAR], [VOSS], [KYNG]. Closely related to these Petri-net oriented approaches is the Graph Model of Behavior, which forms part of the SARA design methodology being developed at UCLA [EST], [RAZ].

Recently, methods for the specification and verification of protocols have been extensively studied [SUN79], [SUN82]. Some of these methods are applicable to the more general problem of a specification methodology for digital highly-concurrent systems.

An extensive literature is presently available on the design and implementation of multi-microcomputer systems (for an annotated bibliography see [SAT]). However, most of the papers describe selected aspects of particular, experimental systems. On the other hand, valuable contributions towards a systematic design methodology are [WEI], [VAL-COU], [CAM-ROS], [EST], [KER]. Specific issues relevant to a systematic design methodology are discussed in e.g. [AND-JEN], [ADA-ROL], [LAM], [LYN-FI].

3. TOWARDS A STRUCTURED REQUIREMENTS SPECIFICATION METHODOLOGY

3.1 Main Features and Advantages of Specification Methodology

In this and the following three sections we describe a system requirements specification method which has the following features.

(a) It uses extended net concepts to provide a concise and mathematically precise model.

(b) It introduces a clear separation between control structure and data (processing) structure.

(c) It is based on a structured approach to parallel programming.

In view of the above features the specification method facilitates analysis, design, implementation, verification and testing of the overall system.

3.2 Control/Data Decomposition

The digital systems we are concerned with may be considered as consisting of two parts: a control structure and a data structure [BRU-ALT], [YOE-BRZ], [LEW], [VAL-COU]. The data structure consists of specific devices (operational units) such as adders, counters, etc. The control structure supervises the activities and sequencing of these devices.

Another essential feature of the digital systems we are interested in, is their high degree of concurrency. Furthermore, we assume the devices to operate asynchronously. The combined effect of concurrency and asynchronous operations may be utilized in order to achieve high-speed overall performance of the system.

3.3 Some Basic Control Structures

We first consider a few simple control structures, as well as methods for using them to form more complex structures. As will become evident in the sequel, our approach is strongly related to basic aspects of structured programming.

We shall use Figure 1 to explain some basic concepts, as well as to introduce our first example of a simple control structure.

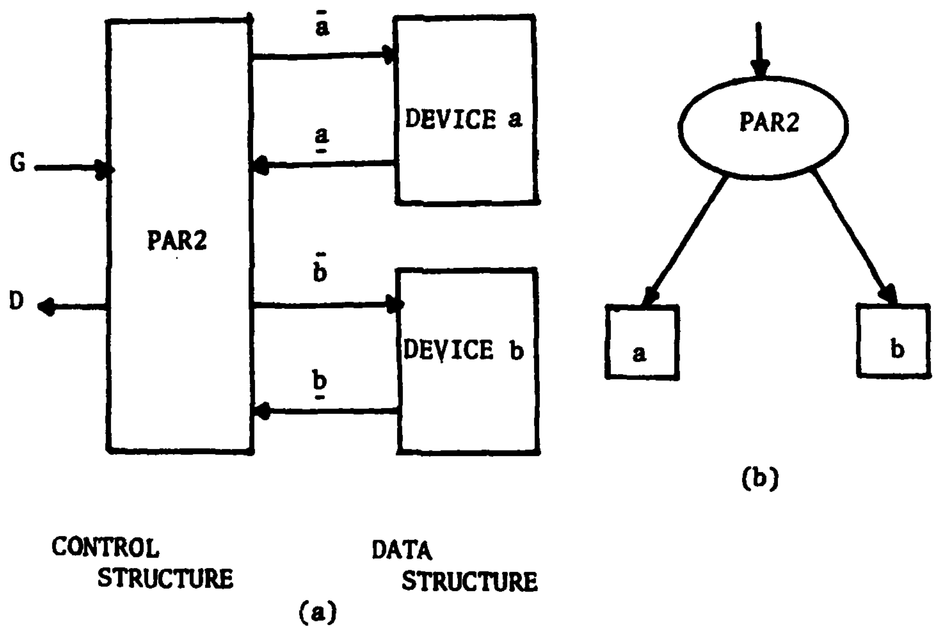


Figure 1. (a) Outside connections of PAR2 control structure
(b) Abbreviated notation.

All the signals indicated in Figure 1 are instantaneous; they may correspond e.g. to the rising edge ($0 \rightarrow 1$ transition) of suitable pulse signals.

Assume the ystem shown in Figure 1 to be idle. Upon the arrival - G "Go") input, the control structure PAR2 becomes active,

by issuing the signals $\bar{a}$ and $\bar{b}$ either concurrently, or one after the other. These signals initiate the operation of the corresponding devices. Each device issues, upon completion of its operation, the corresponding completion signal ($\underline{a}$ or $\underline{b}$). The control structure PAR2 awaits the arrival of both completion signals $\underline{a}$ and $\underline{b}$, whereupon it produces the output D ("Done") and returns to its idle state.

Thus the sequence of signals $G\bar{b}\bar{a}\underline{a}\underline{b}D$ is an example of a feasible input-output sequence which takes the control structure PAR2 exactly once through the cycle of states idle-active-idle. We call any such input-output sequence a <u>basic behavior sequence</u> and denote by B(CS) the <u>basic behavior</u>, i.e. the set of all basic behavior sequences, of the control structure CS. For the control structure PAR2 of Figure 1 we obtain:

$$B(PAR2) = \{G\bar{a}\bar{b}\underline{a}\underline{b}D,\ G\bar{a}\bar{b}\underline{b}\underline{a}D,\ G\bar{b}\bar{a}\underline{a}\underline{b}d,\ G\bar{b}\bar{a}\underline{b}\underline{a}D,\ G\bar{a}\underline{a}\bar{b}\underline{b}D,\ G\bar{b}\underline{b}\bar{a}\underline{a}D\}.$$

Two points concerning this definition of basic behavior need clarification. Firstly, we replace the simultaneous occurrence of two or more signals by their sequential occurrences, in all possible orders. Since we assume all signals to be instantaneous, this approach is well motivated and is closely related to the "Single-Observer Principle" in [MIL], as well as the "Arbitration Condition" in [KEL74]. Secondly, we make no assumptions as to the relative speeds of the control structure and the devices. Hence, we consider e.g. the input-output sequence $G\bar{a}\underline{a}\bar{b}\underline{b}D$ feasible. Namely, we admit the possibility that the completion signal $\underline{a}$ is received before the initiation signal $\bar{b}$ has been produced.

The above expression for B(PAR2) can be simplified by means of the formal language operators introduced in Appendix A. Indeed,

$$B(PAR2) = G \circ (\bar{a}\underline{a} \| \bar{b}\underline{b}) \circ D\ .$$

The preceding considerations are easily extended to a control structure PARk, controlling the concurrent operation of $k \geq 2$ devices. We denote by $\bar{a}_i$ the initiation signal of the i-th device, and by $\underline{a}_i$ its completion signal. Then (see Appendix A)

$$B(\text{PAR}k) = G \circ \| / \{\bar{a}_i \underline{a}_i \mid 1 \leq i \leq k\} \circ D \ .$$

One easily sees, at least informally, that a PAR3 control structure can be obtained by interconnecting two PAR2 structures, as indicated in Figure 2.

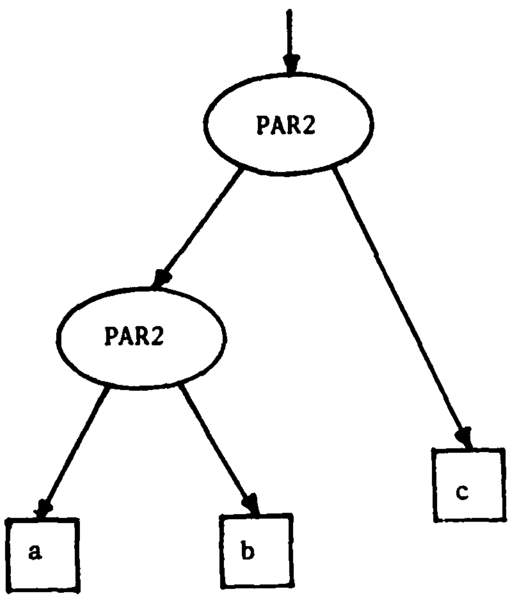


Figure 2. Two PAR2 control structures interconnected to form a PAR3 control structure.

Another simple control structure is SEQk, $k \geq 2$. SEQk activates k devices sequentially (a_1 first, a_k last). Its outside connections are the same as those of PARk, and its basic behavior is specified by

$$B(\text{SEQ}k) = G\bar{a}_1\underline{a}_1 \ \ldots \ \bar{a}_k\underline{a}_k\urcorner$$
$$= G \circ \left(\prod_{i=1}^{k} \bar{a}_i \underline{a}_i \right) \circ L \ .$$

From a purely logic viewpoint, SEQk can be simply realized by connecting corresponding parts, namely $G \rightarrow \bar{a}_1$, $\underline{a}_1 \rightarrow \bar{a}_2, \ldots, \underline{a}_{k-1} \rightarrow \bar{a}_k$, $\underline{a}_k \rightarrow D$. From a circuit viewpoint, however, signal regeneration might be necessary. The abbreviated notation for SEQk is shown in Figure 3.

Generally speaking, we assume that the data structure provides status information to the control structure, by means of suitable level-type <u>status signals</u>.

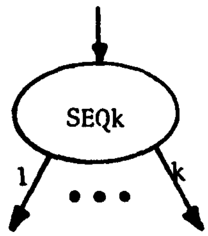
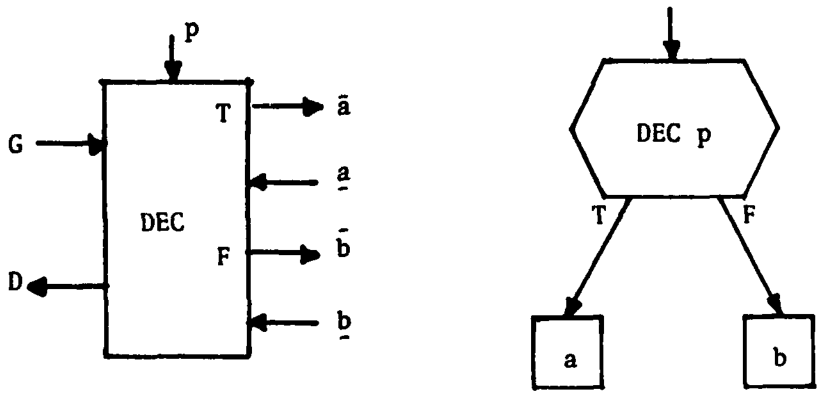


Figure 3. Abbreviated notation for SEQk.

The DEC control structure shown in Figure 4 corresponds to the if-then-else construct of conventional programs.



(a) (b)

Figure 4. (a) Outside connections of DEC control structure
(b) Abbreviated notation.

In Figure 4 we denote by p an incoming (level-type) status signal. We write $\bar{p}$ (instead of $\sim p$ or $\neg p$) to indicate NOT-p.

The basic behavior of the DEC control structure (Figure 4) is then specified by

$B(\mathrm{DEC}) = \{Gp\bar{a}\underline{a}D,\ G\bar{p}\bar{b}\underline{b}D\}$.

Another control structure taken over from conventional (structured) programming is the WHILE structure shown in Figure 5.

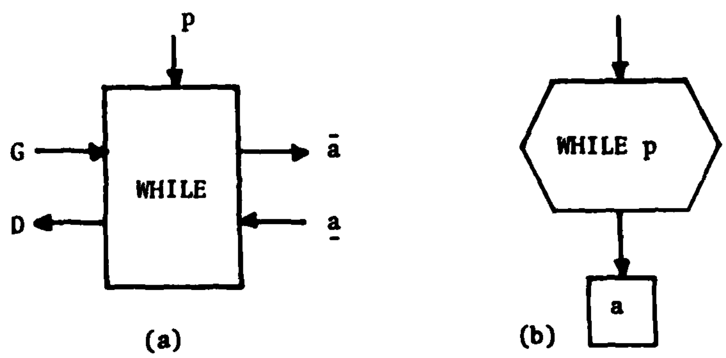


Figure 5. (a) WHILE control structure
(b) Abbreviated notation.

The basic behavior of the WHILE structure of Figure 5 is given by

$B(\mathrm{WHILE}) = G(p\bar{a}\underline{a})^* \bar{p}D$.

Figure 6 shows an example of a parallel computation structure, which illustrates the application of a composite control structure. One easily verifies that for an integer $y \geq 0$ and an arbitrary integer x the computation structure of Figure 6 will produce the product of x and y.

So far we have introduced a few basic control structures and have illustrated the possibility of composing them in order to obtain the control part of a parallel computation structure.

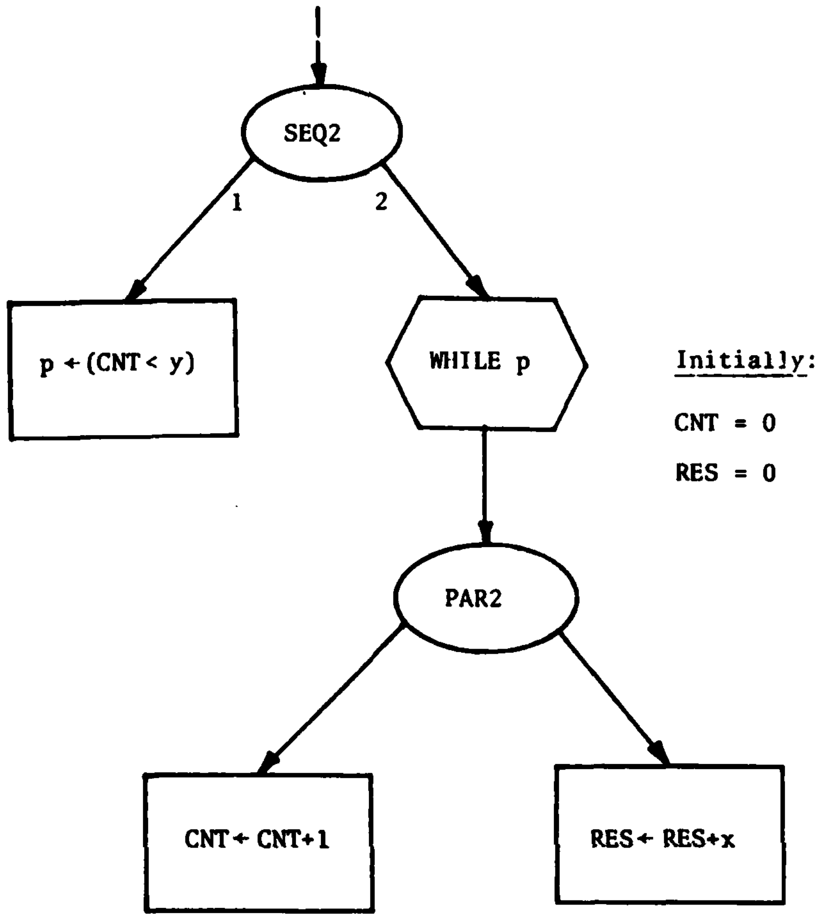


Figure 6. Example of parallel computation structure.

It is noteworthy that the simple control structures introduced so far are quite powerful, when considered as basic building blocks by means of which more complex structures can be composed. Hence these or similar building blocks may be selected as basis for a structured approach to the design of complex control structures (cf. [WEI], [BRU-ALT], [DAC-BLA]).

However, we also wish to investigate control structures which cannot be obtained by the composition of the simple control structures discussed so far. In the sequel we introduce a suitable formalism which will enable us to deal with this problem in a precise and concise way.

4. PARALLEL CONTROL GRAPHS

In this section we introduce the concept of parallel control graph (PCG), following [BOL-YOE] and [YOE-GIN].

4.1 Basic Concepts

Definition 4.1 A parallel control graph (PCG) is a finite, directed graph G with the following properties:

(1) Each node of G is of one of the seven types shown in Figure 7.

(2) Multiple edges are not admitted.

(3) G has exactly one START node S and exactly one HALT node H.

(4) There exists a directed path from S to every other node v of G.

(5) There exists a directed path from every node $v \neq H$ of G to the node H.

Evidently a PCG cannot have self-loops (i.e. cycles of length 1). Examples of PCGs are shown in Figure 8.

We shall refer to nodes of type FORK, JOIN, DECIDER, and UNION as control nodes. A PCG with DECIDER and UNION nodes as only control nodes is purely sequential. Similarly, a PCG with FORK and JOIN nodes as only control nodes is purely parallel.

Definition 4.2 Let G be a PCG. A marking m of G is a function $m: E \to \omega$, where E is the edge set of G and ω is the set of non-negative integers. A marked PCG is an ordered pair (G,m), where G is a PCG and m is a marking of G.

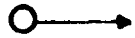
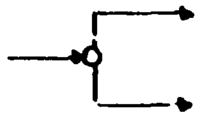
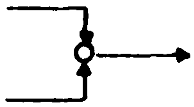
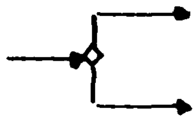
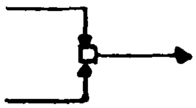
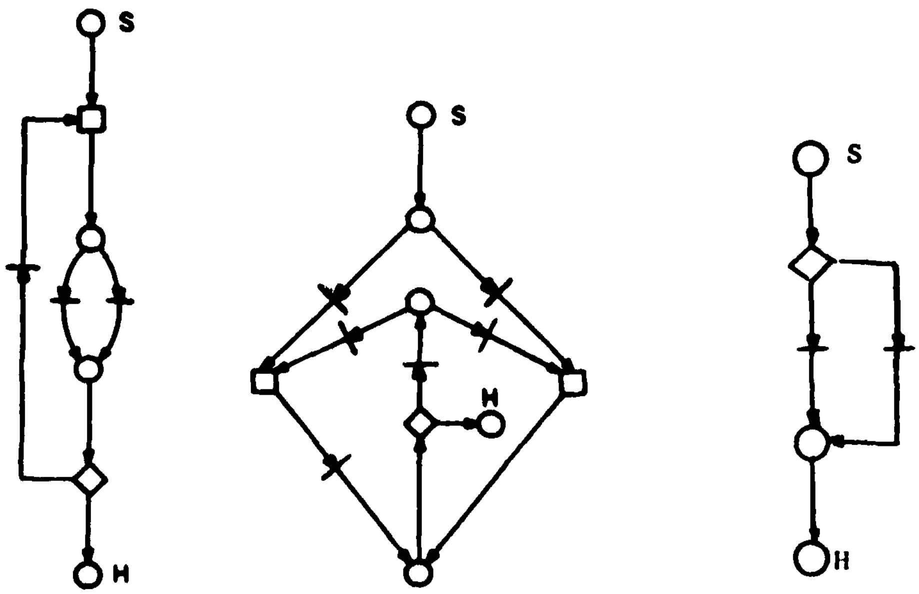
NODE TYPE	INDEGREE	OUTDEGREE	GRAPHICAL REPRESENTATION
START	0	1	
HALT	1	0	
FORK	1	2	
JOIN	2	1	
DECIDER	1	2	
UNION	2	1	
OPERATION	1	1	

Figure 7. Node types of PCG.



(a) (b) (c)

Figure 8. Examples of PCGs.

Let e be an edge of the marked PCG (G,m). We refer to m(e) as the number of tokens on e. If m(e) > 0, we say that e is marked. In the graphical representation of marked PCGs, tokens are indicated by dots (•). Figure 9 shows examples of marked PCGs.

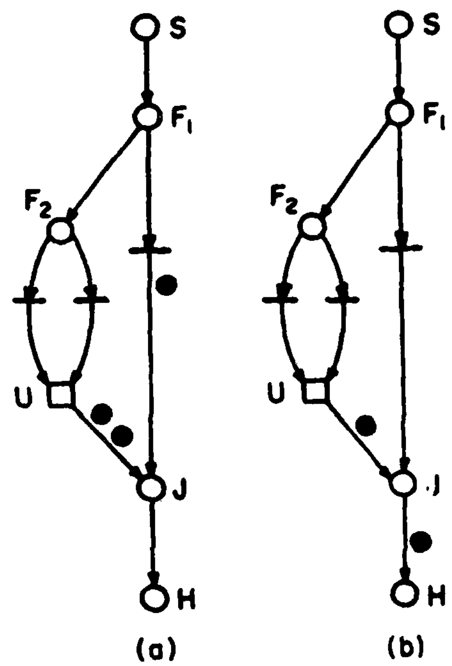


Figure 9. Examples of marked PCGs.

Definition 4.3 Let (G,m) be a marked PCG. A node of type OPERATION or DECIDER or FORK is enabled iff its inedge is marked. A JOIN node is enabled iff both its inedges are marked. A UNION node is enabled iff at least one of its inedges is marked. A node which is enabled may fire.

The firing rules, illustrated in Figure 10, are as follows:

Definition 4.4

(a) The firing of a FORK node decreases the marking of its inedge by 1 and increases the marking of both its outedges by 1.

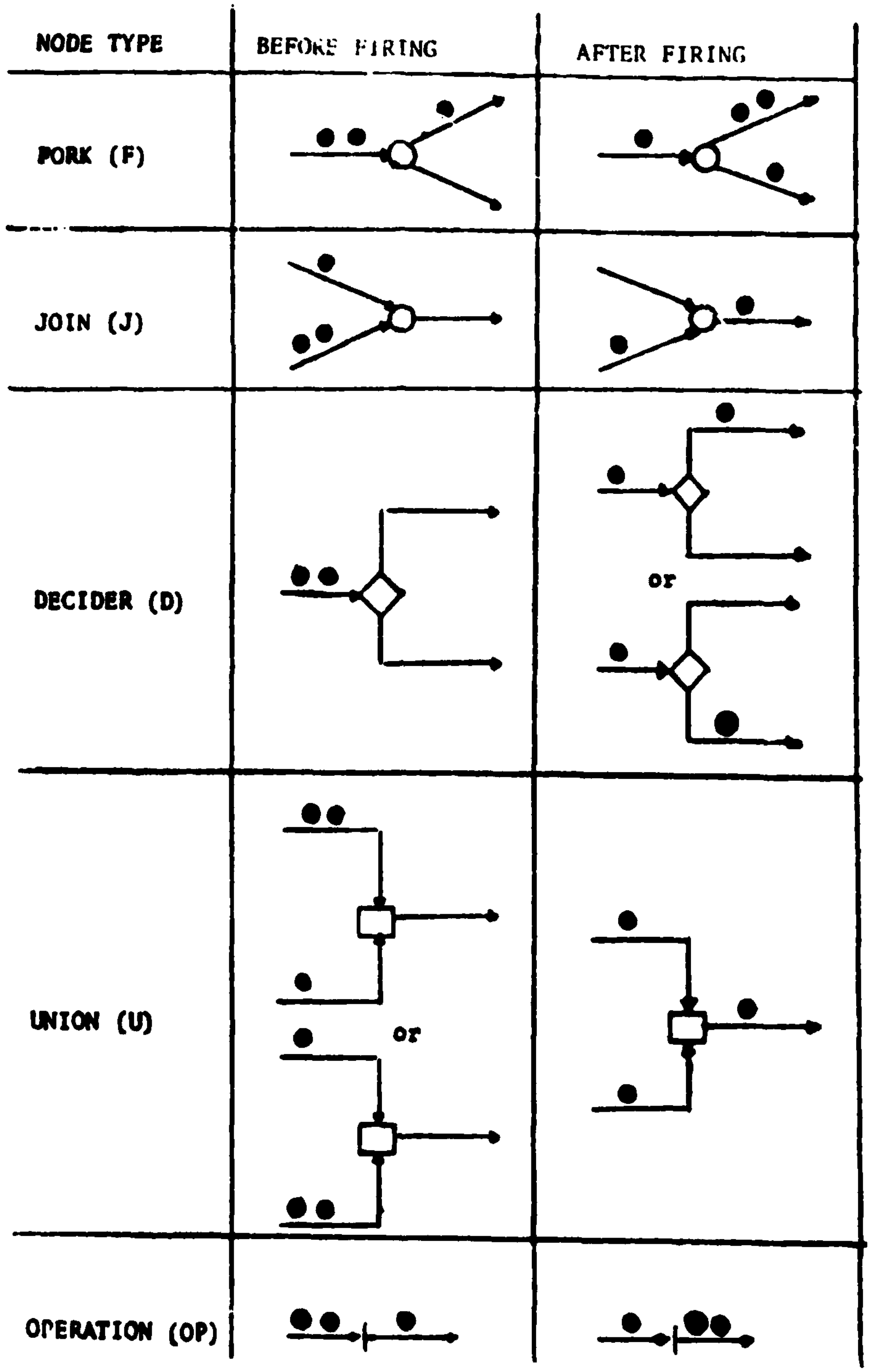


Figure 10. - Examples of "firings"

(b) The firing of a JOIN node decreases the markings of both its inedges by 1, and increases the marking of its outedge by 1.

(c) The firing of a DECIDER node decreases the marking of its inedge by 1, and increases the marking of either one of its outedges by 1.

(d) The firing of a UNION node decreases the marking of one of its marked inedges by 1, and increases the marking of its outedge by 1.

(e) The firing of an OPERATION node decreases the marking of its inedge by 1 and increases the marking of its outedge by 1.

For example, node J in Figure 9(a) is enabled. The firing of J yields the marked PCG of Figure 9(b).

Marked PCGs can, of course, also be defined in terms of Petri nets (cf. [YOE79]).

4.2 Well-Formed PCGs

We now define well-formed PCGs. Let m and m' be markings of the PCG G.

We write $m \overset{v}{\rightarrow} m'$ to indicate that the marking m' is obtainable from the marking m by firing node v. We write $m \rightarrow m'$ to state that m' is reachable from m by the successive firing of one of more nodes of G. Furthermore, we set

$$[m] = \{m' | m \rightarrow m'\} \cup \{m\}.$$

We shall refer to $[m]$ as the set of all markings reachable from m.

We denote by e_S the outedge of the START node S, and by e_H the inedge of the HALT node H.

Definition 4.5 The initial marking m_o of a PCG G is defined as follows:

$$m_o(e_S) = 1 \quad \text{and} \quad m_o(e) = 0 \quad \text{for every} \quad e \neq e_S.$$

A marking m of G is <u>final</u> iff $m(e_H) > 0$. We denote by M_F the set of all final markings of G.

Let G be the PCG shown in Figure 9, m_a its marking shown in Figure 9(a) and m_b the marking shown in Figure 9(b). Then $m_a \in [m_o]$, $m_b \in [m_o]$, and $m_b \in M_F$.

<u>Definition 4.6</u> A PCG G is <u>terminating</u> iff $(\forall m \in [m_o])([m] \cap M_F \neq \emptyset)$ i.e. if m is reachable from m_o, then there exists a final marking reachable from m.

By <u>deadlock</u> we mean a marking m such that $[m] \cap M_F = \emptyset$, i.e. no final marking is reachable from m. Thus, G is terminating iff no deadlock is reachable from m_o.

One easily verifies that the PCGs of Figures 8(a), 8(b) and 9 are terminating, whereas the graph of Figure 8(c) is not terminating.

<u>Definition 4.7</u> Let G be a PCG and E its edge set. G is <u>residue-free</u> iff

$$(\forall m \in [m_o])\left[m \in M_F \rightarrow \sum_{e \in E} m(e) = 1\right],$$

i.e. for any final marking m reachable from m_o, the marked PCG (G,m) contains exactly one token (namely on e_H).

<u>Definition 4.8</u> A PCG G with edge set E is <u>safe</u> iff

$$(\forall m \in [m_o])(\forall e \in E)m(e) \leq 1,$$

i.e. the number of tokens on any edge e cannot exceed 1, under any marking m reachable from m_o.

The following proposition is an immediate consequence of Theorem 3.1 of [YOE-GIN].

<u>Proposition 4.1</u> Every well-formed PCG is safe.

5. PARALLEL CONTROL STRUCTURES

This section is based on [BOL-YOE].

5.1 Basic Concepts

A parallel control structure (PCS) is a suitably labelled PCG [YOE79].

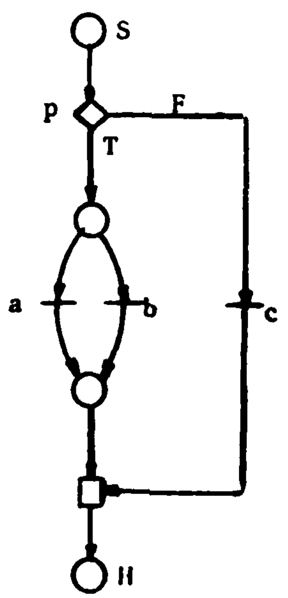
Definition 5.1 A parallel control structure (PCS) Γ consists of the following:

(1) A PCG $G(\Gamma)$

(2) A finite alphabet Σ of operation letters. Every OPERATION node of $G(\Gamma)$ is labelled by a letter of Σ.

(3) A finite alphabet Π of predicate letters. Every DECIDER node D of $G(\Gamma)$ is labelled by a letter of Π. Furthermore, one outgoing edge of D is labelled T (true), and the other edge F (false).

An example of a PCS is shown in Figure 11.



$\Sigma = \{a,b,c\}$

$\Pi = \{p\}$

Figure 11. Example of PCS (Γ_1).

A PCS Γ is well-formed iff $G(\Gamma)$ is well-formed.

<u>Definition 5.2</u> Let G be a PCG. A node sequence

$$(v_1, v_2, \ldots, v_n)$$

is a <u>firing sequence</u> of G iff there exist markings $(m_1, m_2, \ldots, m_n)$ of G such that:

$$m_{i-1} \xrightarrow{v_i} m_i \quad \text{for} \quad 1 \leqslant i \leqslant n,$$

where m_o is the initial marking of G and m_n is final (i.e. $m_n \in M_F$).

<u>Definition 5.3</u> Let Γ be a PCS. We denote by $\bar{\Pi}$ the set of <u>negated</u> predicate letters, i.e.

$$\bar{\Pi} = \{\bar{p} \mid p \in \Pi\}.$$

Let $\alpha = (v_1, v_2, \ldots, v_n)$ be a firing sequence of $G(\Gamma)$ and $(m_1, m_2, \ldots, m_n)$ the corresponding sequence of markings. We associate with every v_i in α a symbol $\tilde{v}_i$ in $\tilde{\Sigma} \cup \{\lambda\}$, where $\tilde{\Sigma} = \Sigma \cup \Pi \cup \bar{\Pi}$ and λ denotes the empty sequence, in accordance with the following rules:

(a) if v_i is a FORK or a JOIN or a UNION, then $\tilde{v}_i = \lambda$.

(b) if v_i is an OPERATION node, then $\tilde{v}_i = \sigma$, where $\sigma \in \Sigma$ is the label of v_i in Γ.

(c) if v_i is a DECIDER with label $p \in \Pi$, outedge e_1 labelled T and outedge e_2 labelled F, then $\tilde{v}_i = p$ if $m_i(e_1) = m_{i-1}(e_1) + 1$, else $\tilde{v}_i = \bar{p}$.

We set $\tilde{\alpha} = \tilde{v}_1 \tilde{v}_2 \ldots \tilde{v}_n$. Thus $\tilde{\alpha} \in (\tilde{\Sigma})^*$.

Definition 5.4 Let Γ be a PCS. With Γ we associate the language $L(\Gamma) \subseteq (\tilde{\Sigma})^*$ defined as follows:

$$L(\Gamma) = \{\tilde{\alpha} \mid \alpha \text{ is a firing sequence of } G(\Gamma)\}.$$

For example, for the PCS Γ_1 of Figure 11 we have

$$L(\Gamma_1) = \{pab, pba, \bar{p}c\}.$$

If $L(\Gamma) = L(\Gamma')$, Γ and Γ' are said to be L-equivalent.

Proposition 5.1 Let Γ be a well-formed PCS. Then $L(\Gamma)$ is regular.

Proof This follows from Proposition 4.1, stating that every well-formed PCG is safe. Thus the set of markings reachable from the initial marking is finite. Hence, there exists a finite automaton A such that $L(A) = L(\Gamma)$. □

Any well-formed PCS Γ represents a control structure CS (see Section 3) in the following sense. Let $\hat{\Gamma}$ be the PCS obtained from Γ by replacing each OPERATION node labelled σ by a sequence of two OPERATION nodes, the first labelled $\bar{\sigma}$ and the second labelled $\underline{\sigma}$. Then

$$B(CS) = G \circ L(\hat{\Gamma}) \circ D.$$

5.2 Composition of PCSs

Structured programs are obtained by "successive composition", using a given set of basic ("primitive") control structures [LE-MAR]. In the following definition we extend this concept of "composition" to PCGs (cf. [YOE79]).

__Definition 5.5__ Let G_1 and G_2 be disjoint PCGs and v an OPERATION node of G_1. We define the __composition__ $G_1(v \leftarrow G_2)$ to be the PCG G obtained by substituting G_2 for v in G_1, as indicated in Figure 12.

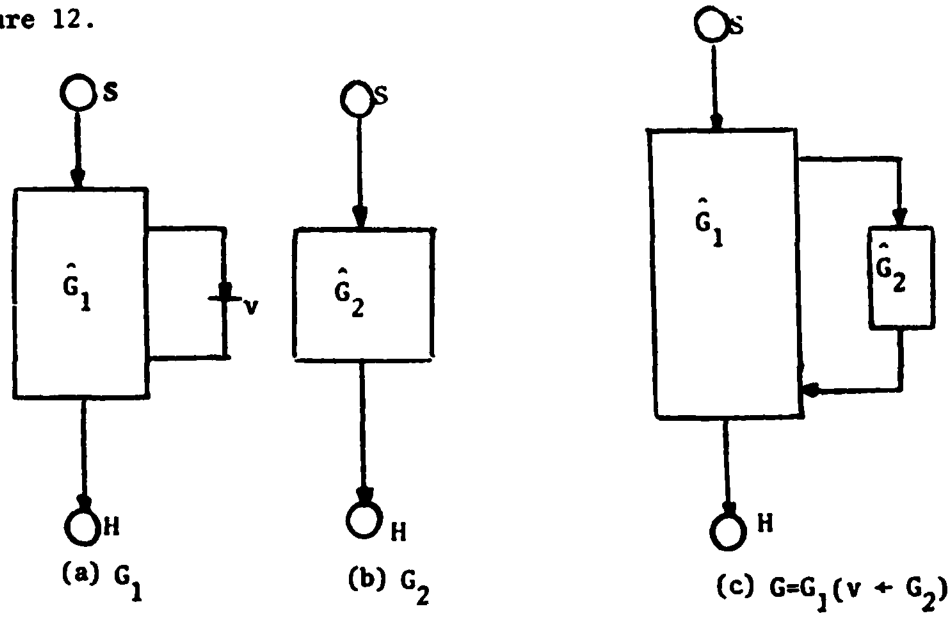


__Figure 12.__ Illustrating the concept of composition
(a) PCG G_1,
(b) PCG G_2,
(c) Composition $G=G_1(v \leftarrow G_2)$.

One easily verifies the following (see [BOL-YOE]).

__Proposition 5.2__ Let G_1 and G_2 be disjoint PCGs, and v an OPERATION node of G_1. Then their composition $G = G_1(v \leftarrow G_2)$ is well-formed iff G_1 and G_2 are well-formed.

The concept of "reducibility" plays an important role in the theory of structured programming (cf. [LE-MAR]).

__Definition 5.6__ Let Δ be a set of well-formed PCGs, $\Delta = \{G_1, G_2, \ldots\}$, and Γ a PCS. Γ is __reducible with respect to__ Δ iff there exists a PCS Γ', such that

(1) $L(\Gamma') = L(\Gamma)$

(2) $G(\Gamma')$ can be obtained by successive compositions of PCGs in Δ.

Figure 13 shows primitive "D-structures" (D for Dijkstra, see [LE-MAR]). D_1 corresponds to SEQ2 (Section 3) in a rather evident way. Similarly, D_2 corresponds to the DEC control structure defined in Section 3.

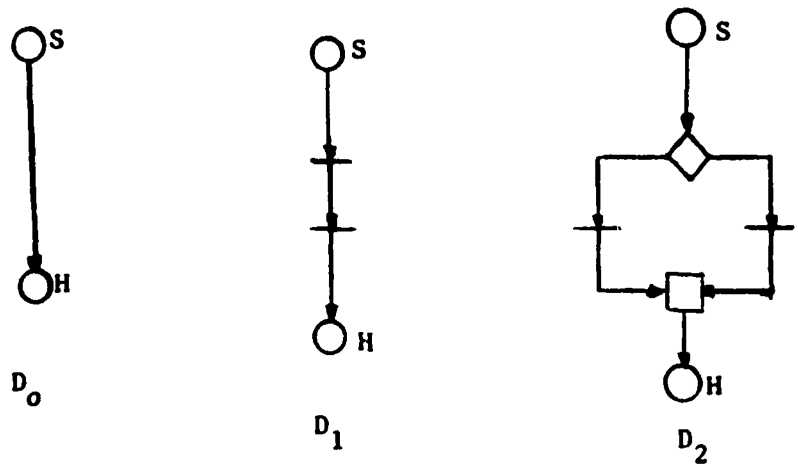


Figure 13. Primitive, cycle-free D-structures

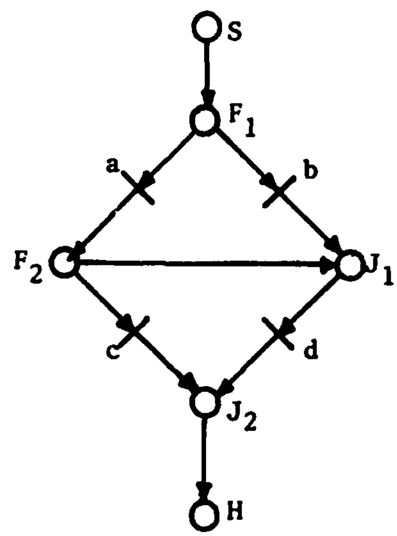
The following proposition is proven in [BOL-YOE].

Proposition 5.3 Let Γ be a well-formed, cycle-free, purely sequential PCS. Then Γ is reducible w.r.t. $\{D_o, D_1, D_2\}$, where the D_i's are shown in Figure 13.

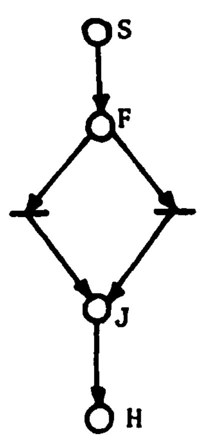
The reducibility of purely parallel PCSs is studied extensively in [GIN-YOE].

Proposition 5.4 The PCS C_2 shown in Figure 14 is irreducible with respect to $\Delta = \{H_1, H_2\}$, where $H_1 = D_1$ (see Figure 13) and H_2 is shown in Figure 15.

<u>Proposition 5.5</u> The PCS C_3 shown in Figure 16 is irreducible with respect to any set Δ of purely parallel PCGs, each having less OPERATION nodes than C_3.



<u>Figure 14</u>. PCS C_2



<u>Figure 15</u>. PCG H_2

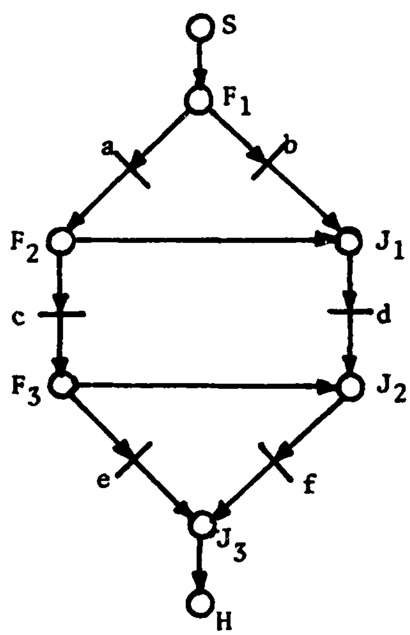


Figure 16. PCS C_3

One easily verifies that the preceding two propositions remain valid even if the corresponding sets Δ are replaced by larger sets $\Delta' = \Delta \cup \Delta_s$, where Δ_s is an arbitrary set of purely sequential PCGs.

The above observations clearly indicate the limitations involved in selecting the simple control structures of Section 3 as a basis for a structured approach to the design of complex control structures. Indeed, the irreducibility results derived in [GIN-YOE] and [BOL-YOE] lead to the establishment of various infinite hierarchies of bases suitable for the structured design of complex PCSs.

6. PARALLEL PROCESSING STRUCTURES

As discussed in Section 3.2, we assume that complex digital systems are composed of two parts: a control structure and a data (processing) structure. The formal concept of PCS, introduced so far, is suitable for modeling the control part of the overall digital system. In this section we define the formal concept of "Parallel Processing Structure" (PPS). Informally, a PPS is derived from a PCS by adding an "interpretation", which represents the data processing part of the system. A PPS is therefore suitable for precisely modeling the overall system. The formal definition of the PPS concept is as follows.

Definition 6.1 A parallel processing structure (PPS) consists of the following:

(1) A PCS Γ (see Definition 5.1)

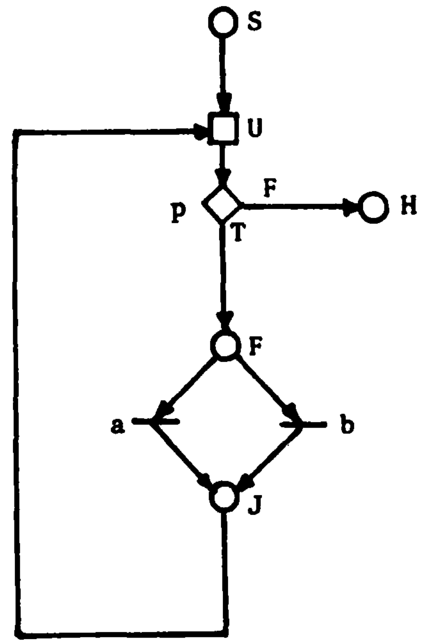
(2) An interpretation I of Γ, $I = (D,A,C)$, where

D is a non-empty set of data (the domain of I);

A is a mapping associating with every operation letter σ of Γ a binary relation $A[\sigma]$ on D, i.e. $A[\sigma] \subseteq D \times D$, in particular, $A[\sigma]: D \to D$ i.e. $A[\sigma]$ is a function.

C is a mapping associating with every predicate letter p of Γ a one-place predicate $C[p]$ on D.

An example of a PPS (Γ, I) is shown in Figure 17. This PPS will perform (similarly to the parallel computation structure of Figure 6) the multiplication $x*y$, for an arbitrary integer x and a nonnegative integer y. The product is obtained as final value of d_2, provided the initial value of $d = (d_1,d_2)$ is set to $(0,0)$.



(a) PCS Γ

domain

$D = \omega \times Z$,

where $\omega \triangleq$ set of nonnegative integers

$Z \triangleq$ set of integers.

Let $d = (d_1,d_2) \in D$, where $d_1 \in \omega$, $d_2 \in Z$;

mapping A

$A[a](d_1,d_2) = (d_1+1,d_2)$

$A[b](d_1,d_2) = (d_1,d_2+x)$, for given integer x ;

mapping C

$C[p](d_1,d_2) \equiv d_i < y$, for given $y \in \omega$.

(b) Interpretation I

Figure 17. Example of a PPS (Γ, I).

Any given PPS performs some computation. This concept will be made precise in the following definition.

<u>Definition 6.2</u> Let (Γ, I) be a PPS. For every operation letter σ of Γ, we set $\hat{\sigma} \triangleq A[\sigma]$. For every predicate letter p of Γ we set

$$\hat{p} \triangleq \{(d,d) | d \in D \wedge C[p](d)\}$$

$$\hat{\bar{p}} \triangleq \{(d,d) | d \in D \wedge \neg C[p](d)\}.$$

Let $w \in \tilde{\Sigma}^*$, where $\tilde{\Sigma} \triangleq \Sigma \cup \Pi \cup \overline{\Pi}$ (see Definition 5.3). With w we associate the binary relation $\hat{w}$ on D as follows.

(1) if $w = \lambda$, then $\hat{w}$ is the identity relation on D, i.e. $\hat{w} = \{(d,d) | d \in D\}$.

(2) if $w = w_1 w_2 \ldots w_r$, where $w_i \in \tilde{\Sigma}$ $(1 \leq i \leq r)$, then $\hat{w} = \hat{w}_1 \cdot \hat{w}_2 \cdot \ldots \cdot \hat{w}_r$, where $\cdot$ denotes composition of binary relations, defined as usual.

The <u>computation performed by</u> the PPS (Γ, I) is the binary relation $C[\Gamma, I]$ on D defined by

$$C[\Gamma, I] \triangleq \cup \{\hat{w} | w \in L(\Gamma)\} .$$

For the PPS (Γ, I) of Figure 17, one easily verifies that

$$(0,0)C[\Gamma, I](d_1, d_2) \text{ implies } d_2 = x*y.$$

7. CONCLUSIONS

7.1 Structured Design of Concurrent Digital Systems

The theory developed in this report provides a suitable framework for a structured, top-down approach to the design of complex, highly concurrent digital systems. It is based on a distinct separation between the control part and the data (processing) part of the system. The control part is modeled by the formal concept of parallel control structure (PCS). It is shown how the well-known methodology of structured programming may be extended to the design of well-formed (particularly deadlock-free) PCSs. An important aspect of any structured approach to design is the selection of suitable, primitive building blocks. The theory of irreducible PCSs, discussed in this report, is therefore an essential contribution to the structured design methodology this report is concerned with.

The overall digital system is modeled by the concept of parallel processing structure (PPS). A PPS consists of a PCS, representing the control part, together with an interpretation, representing the data processing part of the system.

The structured, top-down design of a complex, highly concurrent digital system is best started from a high-level specification in PPS format. This specification is then transformed by stepwise refinements into a low-level description, suitable for direct implementation. Each refinement step can be verified, using well-known techniques of proving parallel programs correct (cf. [KEL76]).

7.2 Proposed Extensions of the PPS Model

The PPS model introduced in this report can easily be extended in order to provide additional modeling power. The incorporation of arbiters as additional building blocks is, no doubt, essential. In [YOE82b] methods were developed for the behavioral specification of arbiters. The formal concepts introduced in [YOE82b] can easily be combined with the PPS model developed so far.

Another important extension of the PPS model consists of the provision of relevant timing information, such as the (minimal and maximal) duration of an operation, maximal delays involved, etc. Similar timing concepts are introduced in [MER] and [MOA-DAV].

7.3 Implementation of a PPS System Description

Assume now that the structured, top-down design methodology summarized in Section 7.1 has led to a low-level PPS description of the required system.

Various techniques are available for the direct, asynchronous hardware implementation of the corresponding PCS. In particular, we refer to [DAC-BLA], [VAL-COU], and [WOJ-CAM]. The data-processing part of the required system, represented by the interpretation of the low-level PPS description, can also be implemented by a variety of techniques. A direct, register-transfer-level approach is discussed in [WOJ-CAM] and [WOJ]. For a VLSI-implementation of the system, the method of implementing a data-path chip described in [MEA-CON] becomes applicable. Alternatively, the data-processing part can be implemented by means of off-the-shelf hardware available for (loosely coupled)

multi-microcomputer systems (cf. [WET]).

As for the direct hardware implementation of arbiters, we refer to [MUE] and [SEI].

8. REFERENCES

[ADA-ROL] G. Adams and T. Rolander, Design Motivations for Multiple Processor Microcomputer Systems. Computer Design, March 1978, pp. 81-89.

[AND-JEN] G.A. Anderson and E.D. Jensen, Computer Interconnection Structures: Taxonomy, Characteristics and Examples. Computing Surveys, Dec. 1975, pp. 197-213.

[BOL-YOE] I. Boldo and M. Yoeli, Reducibility of Parallel Control Structures, TR#283, Computer Science Dept., Technion, Haifa, June 1983.

[BRU-ALT] J. Bruno and S.M. Altman, "A Theory of Asynchronous Control Networks", IEEE Trans. Comp., Vol. C-20, June 1971, pp.629-638.

[CAM-ROS] R. Camposano and W. Rosenstiel, Algorithmische Synthese deterministischer (Petri-) Netze aus Ablaufbeschreibungen digitaler Systeme, Interner Bericht Nr. 22/80, Institut für Informatik IV, University of Karlsruhe, Oct. 1980.

[DAC-BLA] E. Daclin and M. Blanchard, Synthèse des Systèmes Logiques, Cepadues-Editions, 1976.

[DAV] A.M. Davis, "The Design of a Family of Application-Oriented Requirements Languages", Computer, Vol. 15, May 1982, pp. 21-28.

[EST] G. Estrin, "A Methodology for Design of Digital Systems - Supported by SARA at the Age of One", AFIPS Conf. Proc., Nat. Comp. Conf. 1978.

[GIN-YOE] A. Ginzburg and M. Yoeli, Reducibility of Synchronization Structures, 1983, Submitted for publication.

[KEL74] R.M. Keller, "Towards a Theory of Universal Speed-Independent Modules", IEEE Trans. Comp., Vol. C-23, January 1974, pp. 21-33.

[KEL76] R.M. Keller, "Formal Verification of Parallel Programs", Comm. ACM, Vol. 19, 1976, pp. 371-384.

[KER] S. Keramidis, Eine Methode zur Spezifikation und korrekten Implementierung von asynchronen Systemen, Informatik-Arbeitsberichte 15/4, University of Erlangen, June 1982.

[KYNG] M. Kyng, "Specification and Verification of Networks in a Petri-Net Based Language", Proc. 3rd Europ. Workshop on Applications and Theory of Petri Nets, Varenna, Sept. 1982, pp. 302-319.

REFERENCES (cont'd)

[LAM] L. Lamport, "Proving the Correctness of Multiprocess Programs", IEEE Trans. Software Eng., March 1977, pp. 125-143.

[LE-MAR] H.F. Ledgard and M. Marcotty, "A Genealogy of Control Structures", Comm. ACM, Vol.18, 1975, pp. 629-639.

[LEV-MUL] A.A. Levene and G.P. Mullery, "An Investigation of Requirement Specification Languages: Theory and Practice", Computer, Vol.15, May 1982, pp. 50-59.

[LEW] D. Lewin, "Product Specification and Synthesis", Computer Aided Design (ed. G. Musgrave), North-Holland Publ. Comp., Amsterdam, 1979, pp. 25-40.

[LI-MI-WI] R.C. Linger, H.D. Mills and B.I. Witt, Structured Programming: Theory and Practice. Addison-Wesley, 1979.

[LYN-FI] N.A. Lynch and M.J. Fischer, "On Describing the Behavior and Implementation of Distributed Systems", Theoretical Computer Science, Vol. 13, January 1981, pp. 17-43.

[MEA-CON] A. Mead and L.A. Conway, Introduction to VLSI Systems, Addison-Wesley, 1980.

[MER] P. Merlin, "A Methodology for the Design and Implementation of Communication Protocols", IEEE Trans. Comm., Vol. COM-24, June 1976, pp. 614-621.

[MIL] R. Milner, A Calculus of Communicating Systems, Lecture Notes in Computer Science, Vol.92, Springer-Verlag, 1980.

[MOA-DAV] M. Moalla and R. David, "Extension du GRAFCET pur la Representation de Systemes Temps Reel Complexes", Revue RAIRO-Automatique, Vol. 15, No.2, 1981.

[MUE] K. Mühlemann, "Arbiters, Priority Access Conflicts, and the Glitch Problem", Microprocessors and their Applications, Fifth EUROMICRO Symp., August 1979, Goteborg, North-Holland Publ. Co., pp. 391-401.

[QUE] J.P. Queille, The Cesar System: An Aided Design and Certification System for Distributed Applications, RR#214, IMAG, Grenoble, Sept. 1980.

[RAZ] R. Razouk, "Modeling X.25 Using the Graph Model of Behavior," in: [SUN82], pp. 197-214.

REFERENCES (cont'd)

[SEI] C.L. Seitz, "Ideas about Arbiters", LAMBDA, First Quarter 1980, pp. 10-14.

[SUN79] C. Sunshine, "Formal Techniques for Protocol Specification and Verification", Computer, Vol.12, Sept. 1979, pp. 20-27.

[SUN82] C. Sunshine (Ed.), Protocol Specification, Testing, and Verification, North-Holland Publ. Co., 1982.

[VAL-COU] R. Valette and M. Courvoisier, Systemes de Commande en Temps Reel, Editions SCM, Paris 1980.

[VOSS] K. Voss, "Using Predicate/Transition-Nets to Model and Analyze Distributed Database Systems", IEEE Trans. Software Eng., Vol. SE-6, Nov. 1980, pp. 539-544.

[WEI] C. Weitzman, Distributed Micro/Minicomputer Systems, Prentice-Hall, 1980.

[WOJ] H. Wojtkowiak, "Deterministic Systems Design from Functional Specifications", Proc. 18th Design Automation Conference, Nashville, Tennessee, 1981, pp. 98-104.

[WOJ-CAM] H. Wojtkowiak and R. Camposano, "Digital Systems Design with Nets: An Example", Proc. Conf. on Microcomputing, Munich 1979, pp. 135-154.

[YOE79] M. Yoeli, "A Structured Approach to Parallel Programming and Control", Proc. 1st Europ. Conf. on Parallel and Distributed Processing, Toulouse, Febr. 14-16, 1979, pp. 163-169.

[YOE82a] M. Yoeli, "Synthesis of Concurrent Systems", Application and Theory of Petri Nets, eds. C. Girault and W. Reisig, Springer-Verlag, 1982, pp. 183-186.

[YOE82b] M. Yoeli, Behavioral Specifications of Control Structures. Interim Scientific Report No.1, Grant AFOSR 81-0152, June 1982.

[YOE-BAR] M. Yoeli and Z. Barzilai,"Behavioral Descriptions of Communication Switching Systems Using Extended Petri Nets". Digital Processes, Vol. 3, 1977, pp. 307-320.

[YOE-BRZ] M. Yoeli and J.A. Brzozowski, A Model of Parallel Computation Structures. Research Report CS-76-43, Dept. of Computer Science, University of Waterloo, October 1976.

[YOE-GIN] M. Yoeli and A. Ginzburg, "Control Nets for Parallel Processing", Information Processing 80, Ed. S.H. Lavington, North-Holland Publ. Comp., 1980, pp. 71-76.

[ZAV] P. Zave, "An Operational Approach to Requirements Specification for Embedded Systems", IEEE Trans. Software Eng., Vol. SE-8, May 1982, pp. 250-269.

APPENDIX A

BASIC LANGUAGE CONCEPTS

In this Appendix we introduce a few, basic language concepts, used in this Report.

Let Σ denote a finite alphabet. We denote by Σ^* the set of all finite strings (words) of symbols from Σ, including the empty string λ.

A <u>language</u> L <u>over</u> Σ is any subset of Σ^*.

Let L_1 and L_2 be languages over Σ. Their <u>concatenation</u> is defined to be

$$L_1 \circ L_2 = \{xy | x \in L_1 \wedge y \in L_2\},$$

where xy denotes the concatenation of the strings x and y, i.e. string x followed by string y. Usually, we write $x \circ L$ for $\{x\} \circ L$.

$\prod_{i=1}^{k} L_i$ denotes the concatenation $L_1 \circ L_2 \circ \ldots \circ L_k$.

For any language L we set $L^0 = \{\lambda\}$, and $L^n = L^{n-1} \circ L$ for $n \geqslant 1$. Thus $L^1 = L$, $L^2 = L \circ L$, etc. Furthermore, we introduce the usual <u>star</u>-operation $L^* = \bigcup_{i=0}^{\infty} L^i$.

Given $x \in \Sigma^*$ and $y \in \Sigma^*$, the <u>shuffle</u> $x \| y$ is the language over Σ defined recursively as follows:

(1) $\lambda \| \lambda = \{\lambda\}$;

(2) $\sigma \| \lambda = \lambda \| \sigma = \{\sigma\}$, for every $\sigma \in \Sigma$;

(3) Let $\sigma \in \Sigma$, $\tau \in \Sigma$, $x \in \Sigma^*$, $y \in \Sigma^*$.

Then $\sigma x \| \tau y = [\{\sigma\} \circ (x \| \tau y)] \cup [\{\tau\} \circ (\sigma x \| y)]$.

Thus, if $x = \sigma_1\sigma_2 \ldots \sigma_n$, $y = \tau_1\tau_2 \ldots \tau_k$, and $z \in x \| y$, then all σ_i's and τ_j's appear in z exactly once; the relative ordering

of the σ_i's in z is the same as in x, and the relative ordering of the τ_j's in z is the same as in y.

For example, $ab \| cd = \{abcd, acbd, acdb, cabd, cadb, cdab\}$.

For languages L_1 and L_2 over Σ we define their shuffle as:

$$L_1 \| L_2 = \bigcup_{\wedge} \{x \| y \mid x \in L_1,\ y \in L_2\} .$$

Let $\Lambda = \{L_1, L_2, \ldots, L_k\}$, $k \geq 1$ be a finite set of languages over the alphabet Σ. We define the <u>shuffle of</u> Λ (notation: $\|/\Lambda$) as:

$$\|/\Lambda = L_1 \| L_2 \| \ldots \| L_k .$$

APPENDIX B

SUMMARY OF RESEARCH ACTIVITIES SUPPORTED UNDER THIS GRANT

Manuscripts

(1) M. Yoeli and M. Ben-Ari, "Behavioral Description of Arbiters Using Flow Languages", February 1982. Revised version in preparation.

(2) O. Cohen, N. Francez and S. Katz, "Specifying Data Managers - Generalizing Data Structures to Distributed Programming", Extended Abstract, April 1982. Complete version in preparation.

(3) M. Yoeli, "Behavioral Descriptions of Control Structures Using Extended Petri Nets", July 1982. Submitted for publication.

(4) A. Ginzburg and M. Yoeli, "Reducibility of Synchronization Structures", June 1983. Submitted for publication.

Interim Scientific Reports

(1) M. Yoeli, "Behavioral Specifications of Control Structures," June 1982.

(2) G.M. Silberman, "Active Memory Based Tightly-Coupled Multiprocessing Systems", November 1982.

Conferences

M. Yoeli and T. Etzion, "Behavioral Equivalence of Concurrent Systems", presented (by first author) at 3rd European Workshop on Theory and Applications of Petri Nets, Varenna, Italy, Sept. 27-30, 1982. Conference Record, pp. 465 - 478. The paper is also to appear in the selected proceedings of the workshop, to be published by Springer-Verlag.

MISSION
of
Rome Air Development Center

RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C^3I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

ILM